OIPE IAP63
MAR 03 2006
PATENT & TRADEMARK OFFICE

AF

[[40101/06901] CHW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Inventor(s)	:	Darlet
Serial No.	:	09/754,785
Filing Date	:	January 4, 2001
For	:	System and Method for Linear Processing of Software Modules
Group Art Unit:	:	2192
Examiner	:	Eric B. Kiss

Mail Stop: Appeal Brief - Patent
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Certificate of Mailing

I hereby certify that this correspondence is being deposited with U.S. Postal Services as first class mail in an envelope addressed to:

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

By: Date: February 27, 2006
Michael J. Marcin, (Reg. No. 48,198)

TRANSMITTAL

In support to the Notice of Appeal filed December 27, 2005 and the Advisory Action dated February 3, 2006, transmitted herewith please find an Appeal Brief for filing in the above-identified application. Please charge the Credit Card of **Fay Kaplun & Marcin, LLP** in the amount of $500.00 (PTO-Form 2038 is enclosed herewith). The Commissioner is hereby authorized to charge the **Deposit Account of Fay Kaplun & Marcin, LLP NO. 50-1492** for any additional required fees. A copy of this paper is enclosed for that purpose.

Respectfully submitted,

Dated: February 27, 2006

By:
Michael J. Marcin, Reg. 48,198

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

OIPE IAP63
MAR 03 2006
PATENT & TRADEMARK OFFICE

PATENT
Attorney Docket No.: 40101 - 06901

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

In re Application of:	)	
	)	
Pierre-Alain Darlet	)	
	)	
Serial No.: 09/754,785	)	Group Art Unit: 2192
	)	
Filed: January 4, 2001	)	Examiner: Eric B. Kiss
	)	
For: SYSTEM AND METHOD FOR LINEAR PROCESSING OF SOFTWARE MODULES	)	**Board of Patent Appeals and Interferences**

Mail Stop: Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

In support of the Notice of Appeal filed December 27, 2005, and pursuant to 37 C.F.R. § 41.37, Appellant presents in triplicate their appeal brief in the above-captioned application.

This is an appeal to the Board of Patent Appeals and Interferences from the Examiner's final rejection of claims 1-60 in the final Office Action dated August 25, 2005. The appealed claims are set forth in the attached Claims Appendix.

03/06/2006 TTRAN1 00000002 09754785
01 FC:1402 500.00 OP

1. Real Party in Interest

This application is assigned to Wind River Systems, Inc., the real party in interest.

2. Related Appeals and Interferences

There are no other appeals or interferences which would directly affect, be directly affected, or have a bearing on the instant appeal.

3. Status of the Claims

Claims 1-60 have been rejected in the final Office Action. The final rejection of claims 1-60 is being appealed.

4. Status of Amendments

All amendments have been entered in this application.

5. Summary of Claimed Subject Matter

The present invention relates to a system and method for reordering the components of software modules for improved linking process. According to exemplary embodiments of the present invention, a method is described for receiving a software module, wherein the software module include references to locations within the software module and at least some of the references are backward references. (See Specification, p. 5, ll. 16-19). The method also includes reordering components of the software module to remove at least some of

the backward references. (See Id., at p. 5, ll. 25-29).

Further exemplary embodiments of the present invention describe a system that comprises a reorder module. (See Id., p. 13, l. 32 - p. 14, l. 4). The reorder module is configured to receive a software module including references to locations within the software module, wherein at least some of the references are backward references. (See Id.) In addition, the reorder module is configured to reorder components of the software module and remove at least some of the backward references.

Further exemplary embodiments of the present invention also describe a method that comprises receiving a software module sequentially, wherein the software module has at least one symbol reference. (See Id., at p. 20, ll. 7-12). The method further includes linking the software module onto a target memory space, and then resolving the symbol reference(s) without storing the entire software module in local memory while the symbol reference(s) is resolved. (See Id., at p. 21, ll. 7-20).

Further exemplary embodiments of the present invention also describe a system comprising a linker. (See Id., at p. 17, ll. 28-33). The linker is configured to sequentially receive a software module that has at least one symbol reference, then resolve the symbol reference(s), and then store less than the entire software module into local memory during the resolution of the symbol reference(s). (See Id., p. 17, l. 28 - p. 18, l. 3).

Further exemplary embodiments fo the present invention also describe a computer readable storage medium that includes a set of instructions representing a software module that is

executable by a processor. (See Id., at p. 14, ll. 4-7). This set of instructions is operable to receive a software module sequentially, wherein the software module has at least one symbol reference, then link the software module onto a target memory space, and then resolve the symbol reference(s) without storing the entire software module in local memory while the symbol reference(s) is resolved. (See Id.).

Further exemplary embodiments fo the present invention also describe an article of manufacture that comprises a computer-readable medium having stored instructions that are adapted to be executed by a processor. (See Id.). These instructions define a series of steps to be used to reorder a software module when executed. (See Id., at p. 13, l. 21 - p. 14, l. 7). Specifically, the instructions comprise receiving a software module that including references to locations within the software module, wherein at least some of the references being backward references, and then reordering the components of the software module to remove the backward reference(s). (See Id.).

Further exemplary embodiments fo the present invention also describe a further article of manufacture that comprises a computer-readable medium having stored instructions that are adapted to be executed by a processor. (See Id.) These instructions define a series of steps to be used to control the linking of a software module. (See Id., at p. 17, ll. 19-26). Specifically, the instruction comprise receiving a software module sequentially, wherein the software module having at least one symbol reference, then linking the software module onto a target memory space, and then resolving the symbol reference(s) without storing the entire

software module in local memory at one time. (See Id.).

Further exemplary embodiments of the present invention also describe a method that comprises receiving a software module that includes components arranged in a first order. (See Id., at p. 10, l. 28 - p. 11, l. 8). In addition, one of the components includes a reference to a location in a second component, wherein the second component preceding the first component in the first order. (See Id.). The method further includes arranging the components into a second order so that the second component is subsequent to the first component in the second order. (See Id.).

6. Grounds of Rejection to be Reviewed on Appeal

I. Whether claims 36 and 37 are unpatentable under 35 U.S.C. § 101 as being directed to non-statutory subject matter.

II. Whether claims 1-41 and 43- 60 are unpatentable under 35 U.S.C. § 102(b) as being anticipated by John Levine, "Linkers and Loaders, chapter 6," June 1999 [online] accessed 08/15/2005, retrieved from Internet <URL: http://www.iecc.com/linker/linker06.txt>, 9 pages (hereinafter "the Levine reference").

III. Whether claim 42 is unpatentable under 35 U.S.C. § 103(a) as being obvious over the Levine reference in view of U.S. Patent No. 6,185,733 to Breslau, et al. (hereinafter "the Breslau patent").

7. Grouping of Claims

Claims 1-60 may stand or fall together.

8. Argument

I. The Rejection of Claims 36 and 37 Under 35 U.S.C. § 101 as Being Directed to Non-Statutory Subject Matter Should Be Reversed.

A. The Examiner's Rejection

In the final Office Action, the Examiner rejected claims 36 and 37 under 35 U.S.C. § 101 contending that the claimed invention is directed to non-statutory subject matter. (See 08/25/05 Office Action, p. 2, ¶ 4).

B. Claims 36 and 37 Have Been Amended to Recite Statutory Subject Matter.

The Applicant has amended the claims to produce a useful result when fixed in a tangible medium. Specifically, the preamble of claim 36 has been amended to include "A computer readable storage medium including a set of instructions representing a software module that is executable by a processor, the set of instructions..." The storage medium described in

claim 36 would produce a concrete, tangible, and useful result. Since claim 37 is dependent on, and thereby includes all the limitations of, claim 36, it is respectfully submitted that the rejections be withdrawn.

II. The Rejection of Claims 1-41 and 43- 60 are Under 35 U.S.C. § 102(b) as Being Anticipated by the Levine Reference Should Be Reversed.

A. The Examiner's Rejection

In the final Office Action, the Examiner rejected claims 1-41 and 43-60 under 35 U.S.C. § 102(b) as anticipated by the Levine reference. (See 08/25/05 Office Action, p. 3, ¶ 8).

Levine generally describes a technique used by an archive format (i.e., the "ar" command) for creating libraries. A library is handled by a program linker to resolve symbol references within a program. (See the Levine reference, "Library formats," p. 1). The libraries may contain selected object files, or routines, that resolve undefined symbols, and these object files may be used by linkers and loaders in order to automate symbol resolution. (See Id.). In addition, these collections of object files within the libraries may be created using various formats, wherein the simplest format for a library is a sequence of object modules. (See Id.). The format of the linker libraries implemented in UNIX and Windows is an "archive" format, which may be used for collections of any types of files. (See Id.).

For the creation of libraries, the archive formats may use a variety of techniques depending on the support provided by a given operating system. (See Id. at "Creating libraries,"

p. 5). In order to deal with the issue of ordering the object files within a created archive library, UNIX systems contains two programs, "lorder" and "tsort," to help in the creation process. (See Id.). Lorder is used to produce a dependency list of the object files that reference specific symbols in other object files. (See Id.). In other words, Lorder may provide a list of functions that are required by other functions. The creation of the list is accomplished by extracting the symbols using a symbol listing utility, text-processing the symbols, and using standard sort and join utilities to create an output. (See Id.). Tsort performs a topological sort on the output in order to produce a sorted list of object files. (See Id.). Each symbol may be defined after all the references to it, thereby allowing all undefined references to be resolved over a single sequential pass. (See Id.). Therefore, an archive library may be created with the lorder and tsort programs where the output of lorder is used to control the archive library, "ar". (See Id.). Accordingly, lorder and tsort may be used to reorder the dependencies of symbols within an archive library in order to find all external references.

B. The Cited Patents Do Not Disclose Reordering Components of the Software Module to Remove at Least Some of the Backward References as Recited in Claim 1.

The Examiner contends that the Levine reference teaches reordering components of a software module to remove backward references. (See 08/25/05 Office Action, p. 3-4, ¶ 8). However, since the Levine reference simply provides a method for creating an archive library and fails to address the rearrangement of headers, sections, tables, and various other components to

convert of a software module for efficient linking and loading of the software module, the Examiner's claim is unfounded. Thus, it is respectfully submitted that the Levine reference does not teach or disclose "reordering components of the software module to remove at least some of the backward references" as also cited in claim 1.

As described above in the Summary of Claimed Subject Matter, the system and method of the present invention allows for an exemplary software module to be processed into an alternate format by placing various components of the software module in a predetermined order and may eliminate many of the backward references within the software module. Thus, the processed software module may be linked without non-sequentially reading of the software module. Furthermore, instead of saving the entire software module into the linker/loader memory, only a portion of the module may be saved throughout the linking procedure.

The Levine reference provides an explanation for creating an archive library through the use of the Unix command "ar" which combines files into archives (See the Levine reference, "Creating libraries," p. 5). As described above, the use of the lorder and tsort functions with the Levine reference provide nothing more than a means of ordering object files on the basis of symbol dependencies during the creation of an archive library. Lorder is used to extract symbols from the object code. Tsort is used to sort the extracted symbols in the Lorder output. Tsort merely provides a sorted list of symbols and corresponding references. Thus, lorder and tsort are functions used only for symbol resolution during the library creation process. In contrast, the lorder and tsort function do not rearrange the components such as headers,

section, and tables of a software module. The Levine reference does not describe a system of method a placing a section header of an exemplary software module in a more convenient location as to eliminate the need for a link/loader to transition back-and-forth within the software module during the linking process. The sorting, or reordering, performed by the Levine reference is of extracted symbols and just allow for the rearrangement of a symbol directory within an archive library for a linker/loader to reference.

The reordering of the present invention is of specific components of the software module and allows for the linker/loader to read and process the converted software module in a reordered, more convenient manner. The reordering of the extracted symbols is not equivalent to the reordering of software module components. The former allow for improving symbol resolution, while the latter allows for altering a software module for efficient reading and savings during the linking/loading process. Thus, it is respectfully submitted that the Levine reference does not teach or suggest "reordering components of the software module to remove at least some of the backward references," as recited in claim 1.

Therefore, at least for these reasons, it is respectfully submitted that claim 1 is allowable. The Appellant respectfully requests that the Board overturn the Examiner's rejection under 35 U.S.C. § 102(b) of claim 1. Because claims 2-8, and 40-54 depend from, and therefore include all the limitations of, claim 1, Appellant respectfully submits that these claims are allowable and request that the Examiner's rejections of these claims are overturned.

The Examiner rejected claim 9 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 5, ¶ 8). Claim 9 has similar recitations to claim 1. In particular, claim 9 recites "a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module and remove at least some of the backward references." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 1, claim 9 should also be allowed. Therefore, Applicant requests that the rejection of claim 9, and the rejections of claims 10-15, which depend from and include all the limitations of claim 9, be withdrawn.

The Examiner rejected claim 16 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 6, ¶ 8). Claim 16 recites "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." As described above and as described by Levine, the lorder and tsort programs perform functions on object files to define symbols and create an archive library. The process of Levine fails to teach or disclose the method of linking the software module onto a target memory space or using symbol resolution without storing the entire software module in local memory. In fact, Levine fails to suggest any method as to the saving of an amount of the software module. Lorder produces a dependency list of symbol references and tsort produces a sorted dependency list of

symbol references. Levine does not describe the saving of the software module, the amount of the software module saved, nor the location of where the software module is saved. Thus, it is respectfully submitted that Levine's disclosure neither teaches not suggests "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved"as recited in claim 16.

Applicant respectfully submits that for at least the reasons discussed above, claim 16 of the present invention is not anticipated by Levine, and request that the rejection of this claim be withdrawn. Therefore, Applicant requests that the rejection of claim 16, and the rejections of claims 17-22, which depend from and include all the limitations of claim 16, be withdrawn.

The Examiner rejected claim 23 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 6-7, ¶ 8). Claim 23 has similar recitations to claim 16. In particular, claim 23 recites "a the linker configured to resolve the symbol reference, the linker configured to store less than the entire software module in local memory during the resolution of the at least one symbol reference." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 23 should also be allowed. Therefore, Applicant requests that the rejection of claim 23, and the rejections of claims 24-35, which depend from and include all the limitations of claim 9, be withdrawn.

The Examiner rejected claim 36 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 7-8, ¶ 8). Claim 36 has been amended to disclose similar recitations to claim 16. In particular, claim 36 recites "receiving a software module sequentially, the software module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 36 should also be allowed. Therefore, Applicant requests that the rejection of claim 36, and the rejection of claim 37, which depend from and include all the limitations of claim 36, be withdrawn.

The Examiner rejected claim 38 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8, ¶ 8). Claim 38 has similar recitations to claim 1. In particular, claim 9 recites "receiving a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module and remove at least some of the backward references." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 1, claim 38 should also be allowed.

The Examiner rejected claim 39 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8-9, ¶ 8). Claim 39 has similar recitations to claim 16. In particular, claim 39 recites "receiving a software module sequentially, the software

module having at least one symbol reference; linking the software module onto a target memory space; and resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved." Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 16, claim 39 should also be allowed.

The Examiner rejected claim 55 as being anticipated by the discussion of creating libraries in Levine. (See Office Action, p. 8-9, ¶ 8). Claim 55 has similar recitations to claim 1. In particular, claim 55 recites "the software module including components arranged in a first order, a first one of the components including a reference to a location in a second one of the components, the second one of the components preceding the first one of the components in the first order, and arranging the components into a second order." As discussed above, Levine fails to teach or describe a software module having a reference that refers to a location within that software module. Instead, Levine describes the tsort program's use of the output of the lorder program as being a dependency list to resolve symbols. Symbol resolution is not the equivalent of arranging the components into a second order. Thus, Applicant respectfully submits that for at least the reasons discussed above with respect to claim 1, claim 55 should also be allowed. Therefore, Applicant requests that the rejection of claim 55, and the rejection of claim 56-60, which depend from and include all the limitations of claim 55, be withdrawn

III. The Rejection of Claim 42 Under 35 U.S.C. § 103(a) as Being Obvious Over the Levine Reference in View of the Breslau Patent Should Be Reversed.

A. The Examiner's Rejection

In the final Office Action, the Examiner rejected claim 42 under 35 U.S.C. § 103(a) as unpatentable over the Levine reference in view of the Breslau patent. (See 08/25/05 Office Action, pp. 10-11, ¶¶ 9-10).

B. The Cited References Do Not Disclose, for at Least the Same Reasons Described Above, the Limitations, as Recited in Claim 42.

As discussed above, the Levine reference does not teach or suggest all the limitations of independent claims 1. The Breslau patent does not cure the defects of the reference Levine. Accordingly, because claim 42 depends from and, therefore, includes all of the limitations of corresponding independent claim 1, it is respectfully submitted that claim 42 is also allowable over the cited references.

The Breslau patent relates to a library search process within a computer linkage editor program. (See the Breslau patent, col. 1, ll. 15-17). The linkage editor performs the editing on the linkage editor statements and the object modules to be linked. (See Id, at col. 3, ll. 50-55). The object modules to be linked may reside in a remote object module or search library or remote system. (See Id, at col. 4, ll. 11-20). However, the Breslau patent does not teach or

suggest a system or method for "reordering components of the software module to remove at least some of the backward references" as recited in claim 1. Therefore, Appellant respectfully submits that the Breslau patent does no cure the above-described deficiencies of the Levine reference. Thus, for the same reasons as described above with regard to claim 1, and because claim 42 depends from, and, therefore includes all of the limitations of claim 1, Appellant respectfully submits that this claim is also allowable and request that the Examiner's rejection of this claim is overturned.

8. Conclusions

For the reasons set forth above, Appellant respectfully requests that the Board reverse the final rejections by the Examiner of claims 36 and 37 under 35 U.S.C. § 101, claims 1-41 and 43-60 under 35 U.S.C. § 102 (b), and claim 42 under 35 U.S.C. § 103(a), and indicate that claims 1-60 are allowable.

Respectfully submitted,

Date: February 27, 2005

By:

Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000
Fax: (212) 619-0276

CLAIMS APPENDIX

1. A method, comprising:

receiving a software module, the software module including references to locations within the software module, at least some of the references being backward references; and

reordering components of the software module to remove at least some of the backward references.

2. The method according to claim 1, further comprising:

adjusting at least one of the references in the software module to reflect the reordering of the components of the software module, so that the at least one of the references remains a reference to the same component, but to the component's new, reordered location, the new, reordered location coming after the at least one reference in the software module.

3. The method according to claim 2,

wherein the software module includes a symbol table, the symbol table including backward references when the reordering of the components of the software module and adjusting the at least one of the references have been completed.

4. The method according to claim 2,

wherein the software module includes a symbol table, the software module including no backward references in locations before the symbol table when the reordering of the components of the software module and adjusting the at least one of the references have been completed.

5. The method according to claim 2,

wherein the software module is a relocatable object code module in ELF format when the reordering the components of the software module and adjusting the at least one of the references have been completed.

6. The method according to claim 5,

wherein, when the software module is received, the software module is a relocatable object code module in ELF format, and

wherein, when the reordering the components of the software module and adjusting the at least one of the references have been completed, the software module includes a symbol table, the symbol table including backward references, and the software module includes no backward references from locations before the symbol table.

7. The method according to claim 1,

wherein the software module comprises at least one segment, each at least one segment comprising at least one section, and

wherein sections in the same segment are contiguously located in the software module when the reordering of the components of the software module has been completed.

8. The method according to claim 1,

wherein, when the software module is received, the software module is a relocatable object code module in ELF format.

9. A system, comprising:

a reorder module configured to receive a software module including references to locations within the software module, at least some of the references being backward references, the reorder module configured to reorder components of the software module and remove at least some of the backward references.

10. The system according to claim 9,

wherein the reorder module is configured to adjust a reference in the software module to reflect the reordering of the components of the module.

11. The system according to claim 9,

wherein the software module includes a symbol table, and

wherein the reorder module is configured not to remove backward references from the symbol table.

12. The system according to claim 9,

wherein the software module includes a symbol table, and

wherein the reorder module is configured to remove all backward references from locations before the symbol table in the reordered software module.

13. The system according to claim 9,

wherein the software module includes at least one segment, each of the at least one segments including at least one section, and the reorder module is configured to locate sections in the same

segment contiguously in the reordered software module.

14. The system according to claim 9,

wherein the software module is a relocatable object code module in ELF format, and the reordered software module is a relocatable object code module in ELF format.

15. The system according to claim 14,

wherein the software module includes a symbol table,

wherein the reorder module is configured to adjust a reference in the software module to reflect the reordering of the components of the module,

wherein the reorder module is configured to remove all backward references from locations before the symbol table, and

wherein the reorder module is configured not to remove backward references from the symbol table.

16. A method, comprising

receiving a software module sequentially, the software module having at least one symbol reference;

linking the software module onto a target memory space; and

resolving the at least one symbol reference without storing the entire software module in local memory while the symbol reference is resolved.

17. The method according to claim 16, further comprising:

storing section identification information in local memory while the at least one symbol reference is resolved,

wherein the software module includes at least one section and the section identification information uniquely identifies said at least one section.

18. The method according to claim 16, further comprising:

storing symbol information in local memory, wherein said symbol information is contained in the software module.

19. The method according to claim 16,

wherein the software module includes a data section, and the data section is not stored in local memory while the at least one symbol reference is resolved.

20. The method according to claim 16,

wherein the software module includes a text section, and the text section is not stored in local memory while the at least one symbol reference is resolved.

21. The method according to claim 16,

wherein the software module is a relocatable object code module in ELF format.

22. The method according to claim 21, further comprising:

storing section identification information in local memory while the at least one symbol reference is resolved, wherein the software module includes at least one section and the section

identification information uniquely identifies said at least one section; and

storing symbol information in local memory while the at least one symbol reference is resolved, wherein the symbol information is contained in the software module,

wherein the software module includes a data section, and the data section is not stored in local memory while the at least one symbol reference is resolved.

23. A system, comprising:

a linker configured to sequentially receive a software module having at least one symbol reference, the linker configured to resolve the symbol reference, the linker configured to store less than the entire software module in local memory during the resolution of the at least one symbol reference.

24. The system according to claim 23,

wherein the linker is configured to store section information in local memory while the linker resolves at least one symbol reference, and

wherein the software module sequentially received by the linker includes at least one section, and

wherein the section identification information stored by the linker uniquely identifies said at least one section.

25. The system according to claim 23,

wherein the linker is configured to store symbol information in local memory while the linker resolves the at least one symbol reference, and

wherein the symbol information is contained in the software module received by the linker.

26. The system according to claim 23,

wherein the software module received by the linker includes a data section, and

wherein the linker is configured not to store the data section in local memory while the linker resolves the at least one symbol reference.

27. The system according to claim 23,

wherein the software module received by the linker includes a text section, and

wherein the linker is configured not to store the text section in local memory while the linker resolves the at least one symbol reference.

28. The system according to claim 23, further comprising:

a system symbol table.

29. The system according to claim 28, wherein

the system symbol table includes a system symbol table entry for the at least one symbol reference, the system symbol table entry including a field indicative of a defining software module which defines the at least one symbol reference.

30. The system according to claim 23, further comprising:

a software module list.

31. The system according to claim 30, wherein the software module list includes

a software module list entry for the software module.

32. The system according to claim 23, further comprising:

a link status information data structure.

33. The system according to claim 32, wherein the link status information data structure includes a link status information data structure entry for the software module.

34. The system according to claim 33, further comprising:

a software module list, the software module list including a software module list entry for the software module; and

a system symbol table, the system symbol table including a system symbol table entry for the at least one symbol reference, the system symbol table entry including a field indicative of a defining software module which define the at least one symbol reference.

35. The system according to claim 23,

wherein the software module received by the linker is a relocatable object code module in ELF format.

36. A computer readable storage medium including a set of instructions representing a software module that is executable by a processor, the set of instructions operable to:

receiving a software module sequentially, the software module having at least one symbol reference;

linking the software module onto a target memory space; and

resolving the at least one symbol reference without storing the entire software module in

local memory while the symbol reference is resolved.

37. The set of instructions according to claim 36,

wherein the software module is in ELF format.

38. An article of manufacture comprising a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to reorder a software module, said steps comprising:

receiving a software module, the software module including references to locations within the software module, at least some of the references being backward references; and

reordering the components of the software module to remove at least some of the backward references.

39. An article of manufacture comprising a computer-readable medium having stored thereon instructions adapted to be executed by a processor, the instructions which, when executed, define a series of steps to be used to control the linking of a software module, said steps comprising:

receiving a software module sequentially, the software module having at least one symbol reference;

linking the software module onto a target memory space; and

resolving the at least one symbol reference without storing the entire software module in local memory at one time.

40. The method of claim 1, wherein

the reordering of the components of the software module is completed prior to linking the software module.

41. The method of claim 40, further comprising:

linking the reordered software module.

42. The method of claim 1, further comprising:

transferring the reordered software module to a different computer system; and linking the reordered software module on the different computer system.

43. The method of claim 1, wherein the reordered components include an ELF data section.

44. The method of claim 1, wherein the reordered components include an ELF code section.

45. The method of claim 1, wherein the reordered components include an ELF header table.

46. The method of claim 1, wherein the reordered components include an ELF entry point table.

47. The method of claim 1, wherein the reference points to a section located prior to the reference in the received software module.

48. The method of claim 47, wherein, after the software module has been reordered, the reference is changed to point at the same section, the section having been relocated to appear after the reference in the reordered software module.

49. The method of claim 1, wherein the reference points to a table located prior to the reference in the received software module.

50. The method of claim 49, wherein, after the software module has been reordered, the reference is changed to point at the same table, the table having been relocated to appear after the reference in the reordered software module.

51. The method of claim 1, wherein the reference points into a section located prior to the reference in the received software module.

52. The method of claim 51, wherein, after the software module has been reordered, the reference points into the same section, the section having been relocated to appear after the reference in the reordered software module.

53. The method of claim 1, wherein the reference points into a table located prior to the reference in the received software module.

54. The method of claim 53, wherein, after the software module has been reordered, the reference is changed to point into the same table, the table having been relocated to appear after the reference in the reordered software module.

55. A method, comprising:

receiving a software module, the software module including components arranged in a first order, a first one of the components including a reference to a location in a second one of the components, the second one of the components preceding the first one of the components in the first order; and

arranging the components into a second order so that the second one of the components is subsequent to the first one of the components in the second order.

56. The method of claim 55, wherein the arranging occurs prior to linking the software module.

57. The method of claim 56, further comprising:

linking the software module without storing the entire software module in local memory.

58. The method of claim 57, wherein the components include an ELF table and an ELF section.

59. The method of claim 58, wherein the order of segments within the ELF section is preserved when the section is moved to a different position in the reordered software module.

60. The method of claim 59, wherein the only backward references between different ELF components in the reordered software module are references located in the ELF symbol table.

EVIDENCE APPENDIX

No evidence has been entered or relied upon in the present appeal.

RELATED PROCEEDING APPENDIX

No decisions have been rendered regarding the present appeal or any proceedings related thereto.